第一章.

一. 1. 管理硬件和软件的一组软件集合.

4. 系统能及时响应外部事件的请求，在规定的时间内完成对该事件的处理，并控制所有实时任务协调一致地运行.

5. 一段时间内只允许一个进程访问该资源.

三. 1. 硬件系统 软件系统.

3. 硬件 软件

5. 共享性 异步性.

第二章.

一. 3. 每个进程中访问临界资源的那段程序.

4. 异步环境下的一组并发进程因直接制约而互相发送消息、互相合作，互相等待，使得各进程按一定的速度执行的过程.

三. 1. 动态性 异步性 制约性.

5. 动态， 静态

6. 间接制约

8. 管道通信 共享存储 消息传递.

9. 执行 就绪 阻塞

五. 2. $S_1 \to S_2$, $S_1 \to S_3$, $S_2 \to S_4$, $S_2 \to S_5$, $S_3 \to S_6$, $S_4 \to S_7$, $S_5 \to S_7$, $S_6 \to S_7$

semaphore a, b, c, d, e, f, g, h = 0, 0, 0, 0, 0, 0, 0, 0

```
S1(){
    处理;
    signal(a);
    signal(b);
}

S2(){
    wait(a);
    处理;
    signal(c);
    signal(d);
}

S3(){
    wait(b);
    处理;
    signal(e);
}

S4(){
    wait(c);
    处理;
    signal(f);
}

S5(){
    wait(d);
    处理;
    signal(g);
}

S6(){
    wait(e)
    处理;
    signal(h);
}

S7(){
    wait(f)
    wait(g)
    wait(h);
    处理;
}

main(){
    Cobegin
    S1(); S2(); S3(); S4();
    S5(); S6(); S7();
    Coend
}
```

3. semaphore plate = 1, orange = 0, apple = 0, mutex = 1.

```
Dad(){
    while(1){
        P(plate);
        P(mutex);
        放水果;
        V(mutex);
        if(水果==橘子) V(orange);
        else V(apple);
    }
}
```

```
Son(){
    while(1){
        P(orange);
        P(mutex);
        吃橘子;
        V(mutex);
        V(plate);
    }
}
```

```
Daughter(){
    while(1){
        P(apple);
        P(mutex);
        吃苹果;
        V(mutex);
        V(plate);
    }
}
```

```
main(){
    cobegin
    Dad(); Son(); Daughter();
    coend
}
```

4. semaphore empty1 = 1, full1 = 0, empty2 = 1, full2 = 0.

```
PA(){
    while(1){
        从磁盘读记录;
        P(empty1);
        放到缓冲区1;
        V(full1);
    }
}
```

```
PB(){
    while(1){
        P(full1);
        取记录;
        V(empty1);
        P(empty2);
        放到缓冲区2中;
        V(full2);
    }
}
```

```
PC(){
    while(1){
        P(full2);
        取记录;
        V(empty2);
        打印;
    }
}
```

```
main(){
    cobegin
    PA(); PB(); PC();
```

5.
```
int count=0
semaphore mutex=1, empty=1, full=0;
semaphore pay=0, receipt=0;
guest(){
    wait(mutex);
    if(count>=N){
        signal(mutex);
        离开理发店;
    }
    else{
        count++;
        signal(mutex);
        在沙发就座;
        wait(empty);
        离开沙发，坐到理发椅上;
        wait(mutex);
        count--;
        signal(mutex);
        signal(full);
        理发;
        付费;
        signal(pay);
        wait(receipt);
        signal(empty);
        离开理发店;
    }
}

Barber(){
    while(1){
        wait(full);
        理发;
        wait(pay);
        收费;
        signal(receipt);
    }
}

main(){
    cobegin
      guest(); Barber();
    coend
}
```

第三章

二、2. 多道程序运行相互争夺处理机资源时，从就绪队列中，按照一定的算法选择一个进程并将处理机分配给它运行，以实现进程并发地执行。

3. 作业从提交到完成所时间间隔。

4. 指一组进程中的每个进程都在等待由该组进程中的其他进程才能引发的事件发生，~~都无法继续~~。

三、2. 提交 挂起 执行 完成

6. 预防死锁 避免死锁 检测死锁 解除死锁

五、1. ① FCFS:

进程	到达时间	服务时间	开始执行	完成时间	周转时间	带权周转
A	0	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	9	13	9	2.25
D	6	5	13	18	12	2.4
E	8	2	18	20	12	6

平均周转：$\frac{3+7+9+12+12}{5}=8.6$

平均带权周转：$\frac{1+1.17+2.25+2.4+6}{5}=2.56$

② 非抢占SJF:

进程	到达时间	服务时间	开始执行	完成时间	周转时间	带权周转
A	0	3	0	3	3	1
B	2	6	3	9	7	1.17
C	4	4	11	15	11	2.75
D	6	5	15	20	14	2.8
E	8	2	9	11	3	1.5

平均周转：$\frac{3+7+11+14+3}{5}=7.6$

平均带权周转：$\frac{1+1.17+2.75+2.8+1.5}{5}=1.84$.

③ 按为SJF：

	A	B	C	D	E	平均
完成时间	3	15	8	20	10	
周转时间	3	13	4	14	2	7.2
带权周转时间	1	2.16	1	2.8	1	1.59

④ HRRN：

	A	B	C	D	E	平均
完成时间：	3	9	13	20	15	
周转时间：	3	7	9	14	7	8
带权周转时间：	1	1.17	2.25	2.8	3.5	2.14

⑤ RR：

	A	B	C	D	E	平均
完成时间：	4	18	17	20	15	
周转时间：	4	16	13	14	7	10.8
带权周转时间：	1.33	2.67	3.25	2.8	3.5	2.7

2.

A	每20ms	10ms
B	每50ms	10ms
C	每50ms	15ms

0ms：A_1：20-10-0=20
B_1：50-10-0=40
C_1：50-15-0=35.
∴ A_1先执行.

10ms：B_1：50-10-10=30
C_1：50-15-10=25
∴ C_1先执行.

20ms：A_2：40-~~10~~-20=10≠0.
B_1：50-10-20=20
C_1：50-15-20=15

25ms：A_2：40-10-25=5
B_1：50-10-25=15
∴ A_2先执行.

35ms：仅B_1执行.

40ms：A_3：60-10-40=10≠0.

45ms：仅A_3执行

50ms：B_2：100-10-50=40≠0
C_2：100-15-50=35≠0.

55ms：B_2：100-10-55=35
C_2：100-15-55=30
∴ C_2先执行

60ms：A_{B4}：60-10-60=10≠0.

…ms: A4: 80−10−70 = 0

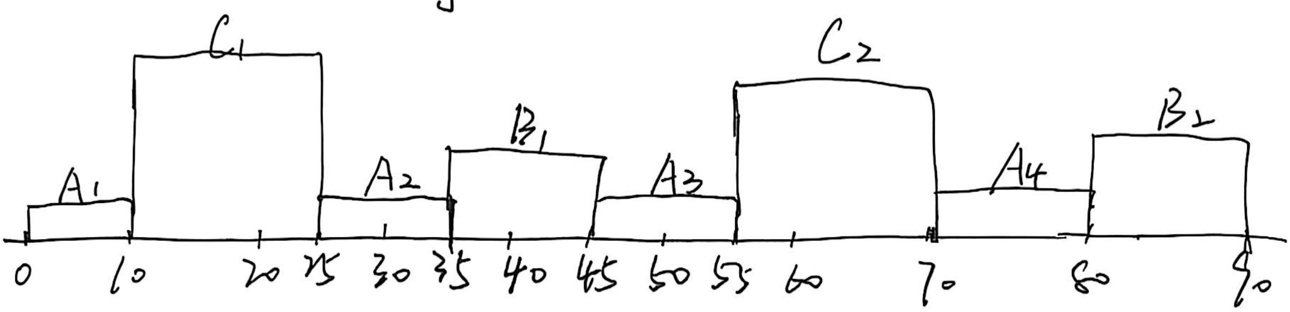
B2: 100−10−70 = 20.

∴ A4 先执行.

…0ms: A3: 100−80−10 = 10

B2: 100−10−80 = 10.

B2 先执行.



(1) Allocation = Max − Need.

进程	Allocation			
P0	0	0	3	2
P1	1	0	0	0
P2	1	3	5	4
P3	0	3	3	2
P4	0	0	1	4

利用安全检测算法:

进程	Work	Need	Work + Allocation	finish
P0	1 6 2 2	0 0 1 2	1 6 5 4	true
P3	1 6 5 4	0 6 5 2	1 9 8 6	true
P1	1 9 8 6	1 7 5 0	2 9 8 6	true
P2	2 9 8 6	2 3 5 6	3 12 13 10	true
P4	3 12 13 10	0 6 5 6	3 12 14 14	true.

[illegible]

(2) 利用银行家算法。

① Request2(1,2,2,2) ≤ Need2(2,3,5,6)

② Request2(1,2,2,2) ≤ Available(1,6,2,2)

③ 假定给P2分配(1,2,2,2)

则 Available = (0,4,0,0)

Allocation2 = (2,5,7,6)

Need2 = (1,1,3,4)

④ 进行安全检测，对所有进程，Needi ≤ Available(0,4,0,0)不成立

即不安全状态。∴不能给它分配资源。

(3) 否。